Programación avanzada en Python

Lección 1: Tipos de datos, operadores y expresiones

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Tipos de datos, operadores y expresiones

PRESENTACIÓN Y OBJETIVOS

En este primer capitulo vamos a ver los principales tipos de datos que podemos utilizar en Python viendo algunos ejemplos de usos. Además, aprenderemos a leer recoger datos por pantalla. Por último, veremos los operadores relacionales y lógicos viendo ejemplos de usos para entender su funcionamiento.

Objetivos

- En esta lección aprenderás a:
- Tipos de datos en Python.
- Como trabajar con cadenas y listas
- Utilizar los operadores relaciones y lógicos

1. TIPOS DE DATOS DE PYTHON

Una de las principales razones que hacen a Python diferente y flexible de otros lenguajes de programación es la asignación dinámica de tipos de datos a sus variables. Esto quiere decir que las variables no necesitan definirse inicialmente con un tipo de datos. Cuando se asignan valores a algunas variables sin predefinición, el tipo de datos se asignará automáticamente a la variable en función del valor.

1.1 Números

Los tipos de datos numéricos en Python son:

- Int
- Float
- complejos

int - cuando se asigna un valor entero a una variable, el tipo de dato será int.

float- cuando se asigna un valor flotante a una variable, el tipo de dato será float.

complejo- cuando se asigna un valor complejo a una variable, el tipo de dato

será complejo

En Python, el tipo de dato de una variable se puede comprobar mediante una función incorporada, type(), y el parámetro de la función es el nombre de la variable.

Veamos una serie de ejemplos de lo aprendido hasta ahora.

Asignación de valores a diferentes variables.

```
[1]  a = 67
     b = 3.224
     c = 3 + 4j
```

Comprobación del tipo de variable mediante la función type()

```
[2]  print(type(a))
     print(type(b))
     print(type(c))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

Con los números, las operaciones aritméticas pueden hacerse fácilmente

```
print(56+4)
```

```
60
```

```
a = 45
b = 6
c = a + b
print(c)
```

```
51
```

1.2 Textos

Las cadenas de caracteres se representan mediante comillas simples o dobles

```
name = "xyz"
print("Name:", name)
print("Type:",type(name))
```

```
Name: xyz
Type: <class 'str'>
```

Podemos calcular la longitud de una cadena mediante la función de Python len(). El parámetro de la función es el nombre de la variable que queremos calcular y devuelve el número de caracteres de la cadena correspondiente incluyendo espacios en blanco.

```
len(str3)
```

```
9
```

Las cadenas se pueden convertir a minúsculas y mayúsculas utilizando la función lower () y upper () respectivamente.

```
y = "PYTHON"
x = y.lower()
print(x)
```

```
python
```

```
u = "python"
y = u.upper()
print(y)
```

```
'PYTHON'
```

Utilizando el operador '+' se pueden concatenar dos cadenas.

```
str1 = "hello "
str2 = "xyz"
str3 = str1 + str2

print("String after concatenation:", str3)
```

```
String after concatenation: hello xyz
```

En el caso de las cadenas, "+" significa concatenación. Consideremos una suma aritmética:

```
x = 50
y = "50"

add = x + y
print(add)
```

Cuando se ejecuta el código anterior, se muestra un error como

```
----> 4 add = x + y
      5 add

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Al comprobar los tipos x es int e y es string. Para hacer la suma aritmética entre estos números, el tipo de y debe convertirse en int. Convertimos una cadena en un entero llamando a la función int () :

```
z = int(y)
```

A continuación, probamos de nuevo a realizar la suma:

```
add = x + z
print(add)
```

```
100
```

Comprobemos otra operación de variables con diferentes tipos de datos:

```
a = 100
b = 3.4333
sum = a + b
print(sum)
```

```
103.4333
```

En este caso no hay problema en la ejecución y se obtiene la salida en float. Lo que sucede exactamente es que Python convierte automáticamente el tipo de datos de la variable (int) a un tipo superior (float).

1.3 Índices en string

Las cadenas están indexadas desde cero y se puede acceder a cada carácter especificando el valor del índice entre corchetes con el nombre de la cadena.

```
str1 = "python"
str1[4]
```

```
'o'
```

Hay una función incorporada en Python, index (), que se utiliza para obtener la posición o índice de un carácter particular de una palabra pasada por parámetro.

```
position = str1.index("h")
print("position of h in str1:",position)
```

```
position of h in str1: 3
```

Si la palabra o el carácter no se encuentra en la cadena de caracteres, se lanza una excepción:

```
position = str1.index("z")
```

```
----> 1 position = str1.index("z")
      2 print("position of h in str1:",position)

ValueError: substring not found
```

Hay dos parámetros especificados con la función index (), que son opcionales:

start - establece un índice de inicio para comenzar la búsqueda y su valor por defecto es cero

end - establece un índice final para detener la búsqueda, por defecto es el final de la cadena

Sintaxis:

string_name . index (value, start,end)

```
string = "ab cd ab"
position = string.index("a",4,7)

print(position)
```

```
6
```

En el ejemplo anterior, se muestra la posición de 'a' en la cadena después del índice 4 .

1.4 Listas

La lista es un conjunto de elementos guardados de forma ordenada. Sintaxis:

List_name = [item1, item2, item 3... item n]

No hay límite para el número de elementos y los elementos pueden ser de diferentes tipos de datos. Ejemplo:

```
pet_list = ['dog','cat','rabbit']
```

Cada elemento tiene un índice, que comienza en cero y se puede acceder a los elementos por su posición de índice.

```
print(pet_list[2])
```

```
rabbit
```

La lista puede ser modificada asignando el nuevo elemento a la posición del índice que se quiere modificar. A continuación, en la lista pet_list, se modifica el segundo elemento dog por parrot. El índice del segundo elemento es uno y se asigna como:

```
pet_list[1] = 'parrot'
```

La nueva lista se convierte en:

```
print(pet_list)
```

```
['dog', 'parrot', 'rabbit']
```

También es posible añadir un nuevo elemento a la lista. Para ello se utiliza una función incorporada, append(). Sintaxis:

List_name. append (new item)

En la lista anterior se añade un nuevo elemento como:

```
pet_list.append('cat')
```

Entonces la lista se convierte en

```
print(pet_list)
```

```
['dog', 'parrot', 'rabbit', 'cat']
```

Cualquier elemento puede ser eliminado de la lista usando la palabra clave ***del***

```
del pet_list[2]
```

En el ejemplo anterior, se borra el elemento con índice 2 que es el tercer elemento (rabbit), por lo que la nueva lista será:

```
print(pet_list)
```

```
['dog', 'parrot', 'cat']
```

La lista puede tener valores de cualquier tipo de datos

```
shop_list = ['apple', 10, 'orange', 3.5, 20, 'tomato', 10]
```

Utilizando el índice negativo, podemos extraer los elementos del final de la lista.

```
shop_list[-1]
```

```
'tomato'
```

1.5 Slicing

El slicing es un proceso de selección de un conjunto de la lista dando 2 condiciones:

start - índice inicial de la parte requerida, el índice inicial es inclusivo, es decir, estará presente en la lista

end - índice final de la parte requerida y es excluyente, lo que significa que no estará presente en la lista.

Sintaxis:

List_name [start index: end index]

```
shop_list[4:6]
```

```
[20, 'tomato']
```

Si sólo se da el índice inicial, se mostrarán los elementos restantes a partir del índice inicial.

```
shop_list[1:]
```

```
[10, 'orange', 3.5, 20, 'tomato', 10]
```

Del mismo modo, dar el índice final sólo mostrará los elementos desde el principio hasta el índice final.

```
shop_list[:4]
```

```
['apple', 10, 'orange', 20]
```

La indexación negativa también es posible para el slicing; selecciona los elementos hasta el índice inicial.

```
shop_list[:-2]
```

```
['apple', 10, 'orange', 3.5, 20]
```

1.6 Leer desde pantalla

La función raw_input () se utiliza para leer una cadena de caracteres o números desde teclado.

```
number = raw_input("Enter a number:")
```

Entonces se le pedirá al usuario que introduzca un número:

Todos los valores recogidos son tomados como caracteres, por lo que si queremos trabajar con ellos de forma numérica tendremos que convertirlos a tipo entero:

```
Enter a number:2
```

```
print(type(number))
```

```
<class 'str'>
```

1.7 Operadores relacionales

En Python tenemos un tipo de dato más, es el tipo de datos booleano. Tiene 2 posibles valores: True y False. La salida de las operaciones relacionales serán valores booleanos. Existen principalmente 6 operadores relacionales:

Operador	Nombre	Ejemplo
==	Igual	x == y = False
!=	Distinto	x != y = True
>	Mayor	x > y = False
&lt	Menor	x &lt y = True
>=	Mayor o igual	x >= y = False
&lt=	Menor o igual	x &lt y = True

Veamos algunos ejemplos:

```
a = 5
b = 2

print(a==b)
print(a!=b)
print(a>b)
print(a<b)
print(a>=b)
print(a<=b)
```

```
False
True
True
False
True
False
```

1.8 Operadores lógicos

Los operadores lógicos se utilizan para conectar las operaciones relacionales. Hay 3 operadores lógicos principales:

- And - si ambas operaciones relacionales son verdaderas, devuelve true.
- Or - si cualquiera de las operaciones relacionales es verdadero, entonces devuelve true.
- Not - si la operación es falsa, devuelve true. Si es verdadera, devuelve falso.

Vamos a practicar con el ejemplo anterior:

```
print( a==b and a!=b)
```

```
False
```

```
print( a==b or a!=b)
```

```
True
```

```
print( not a!=b)
```

```
False
```

2. PUNTOS CLAVE

- Los tipos numéricos en Python son 3; int, float, y complejos.
- Las cadenas de caracteres pueden ser tratadas como array.
- Las listas son elementos mutable.
- Los operadores relacionales, nos permiten saber la relación existente entre dos variables
- Los operadores lógicos se utilizan para conectar las operaciones relacionales